Interval Newton Iteration in Multiple Precision for the Univariate Case

Nathalie Revol

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Interval Newton iteration in multiple precision for the univariate case

Nathalie Revol, Projet Arénaire, LIP, École Normale Supérieure de Lyon et

laboratoire ANO, Université des Sciences et Technologies de Lille

No 4334

December 2001

THÈME 2

ISSN 0249-6399 ISRN INRIA/RR--4334--FR+ENG

Interval Newton iteration in multiple precision for the univariate case

Nathalie Revol, Projet Arénaire, LIP, École Normale Supérieure de Lyon et laboratoire ANO, Université des Sciences et Technologies de Lille

Thème 2 — Génie logiciel
et calcul symbolique
Projet Arénaire

Rapport de recherche n ° 4334 — December 2001 — 14 pages

Abstract: In this paper, interval arithmetic using an underlying multiple precision arithmetic is briefly presented. Then interval Newton iteration for solving nonlinear equations is introduced. A new Newton's algorithm based on multiple precision interval arithmetic is given, along with its properties: termination, arbitrary accuracy on the computed zeros, automatic and dynamic adaptation of the precision. Finally some experiments illustrate the behaviour of this method.

Key-words: multiple precision interval arithmetic, interval Newton algorithm, arbitrary accuracy, automatic and dynamic adaptation of the precision.

(Résumé : tsvp)

Ce texte est également disponible sous forme de rapport de recherche du Laboratoire de l'Informatique du Parallélisme, cf. `http://www.ens-lyon.fr/LIP`.

Unit´e de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN (France)
T´el´ephone : 04 76 61 52 00 - International: +33 4 76 61 52 00
T´el´ecopie : 04 76 61 52 52 - International: +33 4 76 61 52 52

Itération de Newton par intervalles en précision multiple, dans le cas à une seule variable

Résumé : Dans ce rapport, l'arithmétique par intervalles utilisant une arithmétique en précision multiple est définie. Ensuite l'algorithme de Newton par intervalles pour résoudre des équations non linéaires est présenté. Un nouvel algorithme de Newton, basé sur cette arithmétique par intervalles en multi-précision, est proposé et ses propriétés sont données : terminaison de l'algorithme, obtention des solutions avec une précision arbitrairement grande, adaptation automatique et dynamique de la précision. Quelques résultats expérimentaux sont présentés afin d'illustrer le comportement de cette méthode.

Mots-clé : arithmétique par intervalles en précision multiple, algorithme de Newton par intervalles, précision arbitraire, adaptation automatique et dynamique de la précision.

1 Introduction

The long going increase of the power of computers, for monoprocessors as well as for multiprocessors, has led to an increase of the volume of computations. What is now required is not only the quantity of computations, but also some guarantees on the quality of the result. Let us just give two recent examples which have been dramatic enough to become publicized. The first one concerns the failure of a Patriot missile to track and intercept a Scud in Saudi Arabia during the Golf War: its control computer operated continuously over 100 hours and the accumulation of rounding errors led to an inaccurate tracking calculation; "the Scud subsequently hit an Army barrack killing 28 Americans." The second one took place at the Vancouver Stock Exchange: in 1982 a new index with a nominal value 1000.00 was introduced and after 2 years its official value had dropped to 524.881, whereas its exact value was 1098.811. This difference can be explained by the biased (downward) rounding mode chosen to perform the index updates (cf. [13]). These two examples can be considered as anecdotal, so let us now consider the theoretical studies of the numerical stability of an algorithm. For the QR factorization of dense matrices, it is classically assumed that the number of elementary operations times the machine epsilon should be less than $1/2$: $n^3\varepsilon < 1/2$ with n the dimension of the matrix and $\varepsilon = 1^+ - 1$ where 1^+ is the first machine number strictly greater than 1. In IEEE double precision this implies that $n < 2.10^5$: this is not satisfied by very large simulations involving millions of variables, for instance parallel computations, *i.e.* the theoretical study of numerical validity does not hold, even if in practice the results may be satisfactory. Indeed, these theoretical studies account for dense matrices and worst case behaviour, whereas on average, errors are less dramatic.

Some possible alternatives to the usual floating-point arithmetic are exact arithmetic, multiple-precision, interval arithmetic... The obvious advantage of the floating-point arithmetic is that an elementary operation is very cheap since it is performed by the hardware, however inaccuracies can be serious and no hint about the quality of a result can be obtained. On the opposite, in exact arithmetic, only exact numbers are manipulated but not necessarily their value, which can be not representable; furthermore, each operation is costly. The multiple precision arithmetic has been introduced in order to mimic the capabilities of floating-point arithmetic while offering a higher precision: the working precision is chosen by the user and every usual function is available. Nevertheless, this arbitrarily high precision only delays the influence of the finite precision on the results accuracy. The main problem is that the quality of the final result is still not known.

Interval arithmetic is designed to provide guaranteed enclosures of every result; it is usually based on floating-point arithmetic in order to benefit from the low cost of operations. The drawback is then the lack of accuracy, not reliability. The main issue is to get tight intervals as results; increasing the computing precision enables to have sharp intervals as inputs and consequently sharp intervals as results. We propose to combine multiple precision arithmetic, in order to have accurate results, with interval arithmetic in order to preserve the reliability of computations.

In this paper we present the implementation of a multiple precision interval Newton (for univariate functions) and some properties that the use of this arithmetic enables us to prove.

The paper is organized as follows. The basic principles of interval arithmetic are recalled in section 2 and we explain how to implement it with an underlying multiple precision arithmetic. Interval Newton method is introduced in section 3 and its multiple precision counterpart in section 4, along with its properties: this algorithm always terminates and provides enclosures of the zeros with arbitrary precision. The precision is automatically and dynamically adapted to suit the computational needs. In section 5 we illustrate the behaviour of this method on some classical examples, before concluding and giving directions for future work.

2 Multiple precision interval arithmetic

2.1 Interval arithmetic

The main principle of interval arithmetic is to replace every real number by an interval enclosing it and whose bounds are representable by the computer [1, 11]. For instance, π can be represented by the interval $[3.14159, 3.14160]$ if 6 significant radix-10 digits are used. Data known with some degree of uncertainty can also be represented, for instance data measured with bounded measurement errors. Interval vectors can be defined as vectors having interval components and interval matrices as matrices having interval components.

In this paragraph we assume that an ideal arithmetic is used. The roundings effects will be explained in §2.2. Arithmetic operations are defined for interval operands; the semantic of an interval operation $\diamond$ between two intervals X and Y is the following, if it is defined and with [ ] denoting the smallest enclosing interval:

$$X \diamond Y = [\{x \diamond y \,/\, x \in X,\ y \in Y\}].$$

For instance

$$\begin{array}{lcl} [-2,3] + [5,7] & = & [3,10], \\ [-3,2] \times [-3,2] & = & [-6,9] \text{ differs from} \\ [-3,2]^2 & = & [0,9], \\ [-3,2]/[0.5,1] & = & [-6,4]. \end{array}$$

The definitions of the usual functions can also be extended to include interval arguments: for instance $\exp[-2,3] = [\exp(-2), \exp(3)]$ since exp is an increasing function, and $\sin[\pi/3, \pi] = [0,1]$, here care must be taken as sin is not monotonous.

The major advantage of this arithmetic is the fact that every result is guaranteed, contrary to usual floating-point arithmetic: if input data are scalars and the computations are performed using intervals, the result of the exact scalar calculation belongs to the resulting interval. Nevertheless, this enclosure can be very large: this is due to two phenomena known as the variable dependency and the wrapping effect. The first one can be observed in

$$X \times (Y + Z) \subset X \times Y + X \times Z$$

with an inclusion instead of the expected equality: indeed,

$$X \times (Y + Z) = \{x \times (y + z), x \in X, y \in Y, z \in Z\}$$

whereas

$$X \times Y + X \times Z = \{x \times y + x' \times z, x \in X, x' \in X, y \in Y, z \in Z\}$$

where the identity of x and x' is lost. The second one comes from the *wrapping* of every result in $\mathbb{R}^n$ into a box with sides parallel to the axes.

Moreover, with interval arithmetic, operations lose their algebraic properties, for instance $-$ is not the reciprocal of $+$: $[-2,3] + [5,7] = [3,10]$, but subtracting $[5,7]$ to $[3,10]$ does not give $[-2,3]$ back, since $[3,10] - [5,7] = [-4,5] \supset [-2,3]$, but $\neq [-2,3]$. This means that when deriving an algorithm, both expressions are not equivalent. Another property which is lost is the distributivity of $*$ over $+$, because of the variable dependency, as shown above.

Let f be a function and I an interval vector, the problem is to determine the smallest interval containing $f(I)$ the range of f over I. It has already been mentioned that different expressions for f, which are mathematically equivalent, do not yield the same result when variables are replaced by intervals. (From now on, a capital letter will refer to one of the interval enclosures of a scalar function.) Usually, these results are much larger than the exact range of f over I. The following example illustrates the fact that the replacement of the scalar variable by the interval I in the expression of f does not give the desired result; furthermore, the result depends on the expression used to define f. If $f : x \mapsto x^2 - 2x + 1$ and $I = [-1,3]$,

$$I^2 - 2 * I + 1 = [-1,3]^2 - 2[-1,3] + 1 = [0,9] + [-6,2] + 1 = [-5,12]$$

and if the expression $f(x) = x * (x-2) + 1$ is used instead,

$$I * (I-2) + 1 = [-1,3] * ([-1,3] - 2) + 1 = [-1,3] * [-3,1] + 1 = [-9,3] + 1 = [-8,4]$$

whereas $F(I) = f(I) = [0,4]$.

Thus a realistic goal is to obtain a quite tight enclosure of the range of a function. If the function is a univariate polynomial, then Horner's evaluation gives a relatively sharp enclosure. Another way to tighten the resulting intervals consists in using Taylor-Lagrange formulas, also known as the mean-value theorem [12]: the first order formula states that if $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable then

$$\forall x,\ \forall y,\ \exists \xi \in [x,y] \ /\ f(y) = f(x) + \mathrm{grad} f(\xi).(y-x).$$

This can be translated for interval evaluation as

$$\forall X \text{ interval},\ \forall x \in X,\ f(X) \subset f(x) + \mathrm{Grad} f(X).(X-x).$$

Similarly, the second order formula has an interval translation (Hf is an interval enclosure of the Hessian of f):

$$\forall X \text{ interval},\ \forall x \in X,\ f(X) \subset f(x) + \mathrm{grad} f(x).(X-x) + \frac{1}{2}(X-x)^t.Hf(X).(X-x).$$

When X is small, then a Taylor-Lagrange formula produces a tight enclosure of $f(X)$. A requirement is that the derivatives of f are available and this can relatively easily be obtained via automatic differentiation.

Since sharp bounds can be obtained for small intervals using Taylor-Lagrange formulas, a naïve idea consists in partitioning the interval I into very small subintervals and in evaluating f over each subinterval, since the following inclusion holds and is often a proper inclusion:

$$\forall I, \forall I_1, I_2 \ / \ I_1 \cup I_2 = I, \ F(I_1) \cup F(I_2) \subset F(I),$$

i.e. by recursively bisecting I, a tight enclosure of $f(I)$ can be obtained as the union of the enclosures of f over the subintervals. The main drawback of this method is that it requires an exponential number of evaluation of f over subintervals and thus it is not used to evaluate a function. However, it is used to solve other problems such as root finding or global optimization of a continuous function.

2.2 Multiple precision interval arithmetic

Interval arithmetic has been defined from a theoretical point of view in §2.1. In order to implement this arithmetic, directed roundings are needed. In floating-point arithmetic, these roundings are available to the programmer if both the floating-point unit and the programming language comply with the IEEE-754 norm, as far as arithmetic and algebraic operations are concerned. For instance, the floating-point representation of $\sqrt{[2,3]}$ is $[\nabla\sqrt{2}, \Delta\sqrt{3}]$ the outward rounded representable interval for $\sqrt{[2,3]}$, where ∇ denotes the downward rounding and Δ the upward rounding. In the IEEE-754 norm, the correct rounding is not required for elementary functions such as exp, log and trigonometric functions, because it is not known yet how to provide it and it is thus quite tedious to implement an interval arithmetic based on IEEE fixed precision floating-point arithmetic. The latter can be replaced by another arithmetic as long as it provides directed roundings or at least enclosures of the result of every operation.

In some experiments on interval global minimization, we encountered the problem of having input intervals bisected until their relative width was ε (defined in section 1) without having reached a satisfactory accuracy on the result. This problem arose with either very "flat valley" functions or "egg-box" functions. With "flat valley" functions, it is impossible to distinguish which point is the lowest in the valley, using the IEEE double precision, and a paving of the bottom of the valley was returned. With an "egg-box" function f, there is a large number of local minima and the values of f at these points are very close: only a highly accurate evaluation of f permits to discard local minima. This led us to work using multiple precision interval arithmetic.

Our library for multiple precision interval arithmetic is MPFI for *Multiple Precision Floating-point Interval arithmetic*, it is a joint development with F. Rouillier and it is available at `http://www.ens-lyon.fr/~nrevol/nr_software.html`. We have chosen the multiple-precision library MPFR (cf. `http://www.mpfr.org`): this library is IEEE-754 compliant, even for the elementary functions, *i.e.* it provides the correct rounding of the result

of every arithmetic operation or elementary function evaluation. Furthermore it is built upon GMP, a library for exact arithmetic known for its efficiency and its portability. Quite simultaneously, the Arithmos library [5] was developed (which is not open-source).

It can be proven that for a large class of functions called "arithmetic expressions" in [11], (which can roughly be described as functions defined without involving the bounds of the interval operands and without tests), the width of a function enclosure is proportional to the width of the input interval if ideal (exact) arithmetic is used:

$$w(F(X)) \leq K_F w(X) \text{ where } K_F \text{ is a constant depending on } F. \tag{1}$$

When replacing this ideal arithmetic by a multiple precision one, if the effect of roundings is also linear in the computing precision then arbitrary accuracy can be reached for the function enclosure. In particular when the function is a polynomial, then it is known that Horner's evaluation satisfy this assumption of linear dependence of the output error to the input error (cf. [8], §5.1). This assumption is also fulfilled when f is a Lipschitz function (in an interval sense) [11]. In order to distinguish between an ideal value Y and the value computed using finite precision arithmetic, let us denote with a hat $\widehat{Y}$ the computed value. For this class of functions, the width of the computed (outward rounded) function enclosure $\widehat{F(X)}$ exceeds the width $w(F(X))$ by a constant times ε, where ε has been defined in section 1 and the constant depends on the function:

$$w(\widehat{F(X)}) \leq w(F(X)) + c_F \varepsilon, \tag{2}$$

i.e. arbitrary accuracy can be reached for the computed function enclosure, if ε can be made as small as required.

3 Interval Newton method

3.1 Classical interval Newton method

Since our implementation deals only with univariate functions, we will concentrate on functions $f : \mathbb{IR} \to \mathbb{IR}$. The main principle of interval Newton method [7, 6, 2, 3, 14] consists in enclosing the graph of f in the cone given by the extremal tangents, cf. the gray zones of figure 1, and in enclosing the zeros in the intersection of this cone and the real axis. Let us denote by X the searching interval and by $\check{x}$ the point defining the cone ($\check{x}$ can be any point of X); the slopes of the limiting straight lines are the extremal values of $F'(X)$ where F' is an enclosure function for f'. For any point (x, y) in the cone, there exists $\alpha \in F'(X)$ such that

$$y = f(\check{x}) + \alpha.(x - \check{x}).$$

For any point in the intersection of the cone with the x-axis, $y = 0$ and thus

$$\exists \alpha \in F'(X) \ : \ x = \check{x} - \frac{f(\check{x})}{\alpha}.$$

Finally X', the intersection of the cone with the x-axis, is defined by

$$X' \subset \check{x} - \frac{f(\check{x})}{F'(X)}$$

and, since the roots are sought in X, the new search interval X'' is defined as

$$X'' := \left(x - \frac{f(x)}{F'(X)} \right) \cap X.$$

If $F'(X)$ contains 0, then an extended interval division is performed and after the intersection with X, two subintervals contained in X are produced, cf. the right part of figure 1.

Let us denote by X_k the previous iterate and by x_k the point defining the cone (in the following $x_k = \mathrm{mid}(X_k)$, the middle of X_k); the new iterate X_{k+1} of interval Newton algorithm is defined as

$$X_{k+1} := (x_k - \frac{f(x_k)}{F'(X_k)}) \cap X_k.$$

The result of the interval Newton method is a list of intervals whose union contains every zero. Every zero appears in one of the intervals of the resulting list, but this list may also contain intervals that do not contain any zero but that could not be discarded since if their interval image by f contains 0.

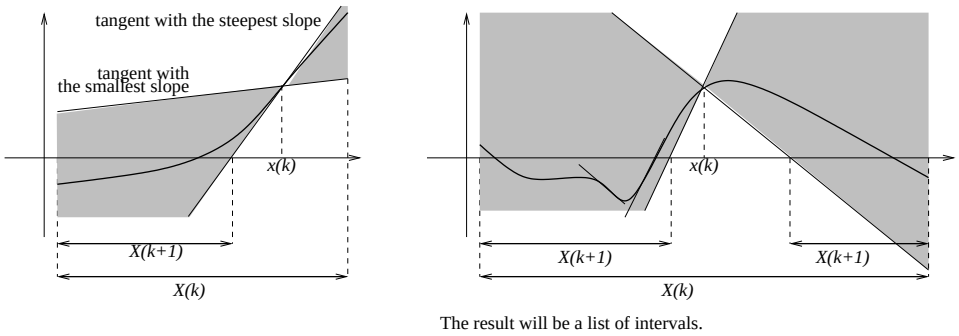

Figure 1: The principle of Newton method.

When the interval arithmetic uses fixed precision floating-point arithmetic to compute the interval bounds, then the stopping criterion on X is:

$$w(X)/|\mathrm{mid}(X)| \leq \varepsilon_X \text{ or } w(F(X)) \leq \varepsilon_Y.$$

Only one of the two conditions needs to be fulfilled, since it may happen that $X = [x^-, x^+]$ with x^- and x^+ two consecutive floating-point numbers whereas $F(X)$ cannot be made smaller than ε_Y (and conversely).

3.2 Properties of the classical interval Newton method

These properties have been established for the interval Newton method and they hold for fixed precision floating-point interval arithmetic as well as for multiple-precision or even exact (theoretical) interval arithmetic. We just recall them in order to be complete.

Proof of existence of a zero

Brouwer's theorem is effective in interval arithmetic: if $x - f(x)/F'(X) \subset X$ then there exists one zero of f in X. Furthermore, since f is univariate, the intermediate value theorem applies: the existence of a zero in $[a, b]$ is ensured if $f(a) \times f(b) < 0$. With interval arithmetic, this can be translated into "let us denote $F([a])$ by $[a^-, a^+]$ and $F([b])$ by $[b^-, b^+]$, there exists a zero of f in $[a, b]$ if $a^+ < 0 < b^-$ or $b^+ < 0 < a^-$".

Proof of existence and uniqueness of a zero

If there is a strict shrink of the search interval during interval Newton iteration, *i.e.* if $x - f(x)/F'(X)$ is strictly contained in the interior of X, then the existence and uniqueness of a zero in X is proven [10].

4 Multiple precision interval Newton method

When an arbitrary precision can be achieved on a function enclosure, then the stopping criterion can be made more stringent. Moreover, in order to ensure the termination of the algorithm, we ensure that the width of the intervals in the queue $\mathcal{L}$ of not yet processed intervals is quickly decreasing. These considerations have led us to propose the following algorithm.

4.1 Multiple precision interval Newton method

Input: f, F', X_0 *the initial search interval*
Initialization: $\mathcal{L} = \{X_0\}$, $\alpha = 0.75$ *(any value in* $]0.5, 1[$ *is suitable)*
Loop: *while* $\mathcal{L} \neq \emptyset$
    Suppress $(X, \mathcal{L})$
    $(X_1, X_2) := (x - \frac{f(x)}{F'(X)}) \cap X$
        (X_2 can be empty, tests on X_2 are performed if $X_2 \neq \emptyset$)
    if $w(X_1) > \alpha w(X)$ *or* $w(X_2) > \alpha w(X)$ *then*
        $(X_1, X_2) :=$ *bisect*(X) *in two halves of equal length*
    Increase the working precision if needed
    if $X_1 \neq \emptyset$ *and* $F(X_1) \ni 0$
        if $w(X_1) \leq \varepsilon_X$ *and* $w(F(X_1)) \leq \varepsilon_Y$ *then*
            Insert (X_1, Res) and test existence and uniqueness
        else
            Insert $(X_1, \mathcal{L})$
    idem for X_2

Output: *Res, a list of intervals that may contain the zeros.*

4.2 Properties

The stopping criterion requires that both the width of the zero enclosure and the function evaluation are small enough: the "OR" of §3.1 is replaced by a "AND" in our algorithm. The termination of the algorithm for arithmetic expressions (cf. [11]) is established by noting that:

- the widths $w(X)$ of the examined intervals X in $\mathcal{L}$ are upper bounded by a geometric series of ratio α. The value of α is set to 0.75 in our algorithm but can be any value in $]0.5, 1[$ (for a discussion on stopping criteria, cf. [4]),
- the widths of the computed function evaluations $w(\widehat{F(X)})$ are upper bounded by a geometric series of ratio α plus the constant term $c_F\varepsilon$ mentioned in equation 2 in §2.2. This second upper bounding geometric series is obtained by observing that $w(F(X)) = \mathcal{O}(w(X))$ (this has been stated in equation 1 in §2.2).

The stopping criterion can be fulfilled if the working precision is sufficient.

A second peculiarity of our algorithm is its automatic and dynamic adaptation of the working precision. The precision is increased when one of the following conditions holds:

- $w(X_1)$ or $w(X_2) \geq w(X)$: the relative width of X equals the current ε and no further refinement is possible using the current precision;
- $w(\widehat{F(X_1)})$ or $w(\widehat{F(X_2)}) \geq w(\widehat{F(X)})$: it implies that $w(F(X)) \leq w(\widehat{F(X_1)}) \leq w(F(X_1)) + c_F\varepsilon \leq w(F(X)) + c_F\varepsilon$, *i.e.* the rounding error is greater than the gain obtained by replacing X by X_1 or X_2 in the computed evaluation by F.

Since Newton's algorithm doubles the accuracy on the zero at each iteration (at least for a single root), we chose to double the current precision when it needs to be increased. In order to minimize the computational cost, the working precision is stored with the interval in the queue $\mathcal{L}$ and is restored when the corresponding interval is examined.

A particularly pleasant feature of Newton's algorithm is that it is auto-corrective: when the precision is increased, only further computations use this higher precision, but the whole computation does not need to be restarted.

5 Experimental results

Our implementation of Newton method presented in §4 deals only with univariate functions for the time being. The multidimensional case requires the solution of a linear system involving the Jacobian and this is not yet implemented for the multiple precision interval arithmetic.

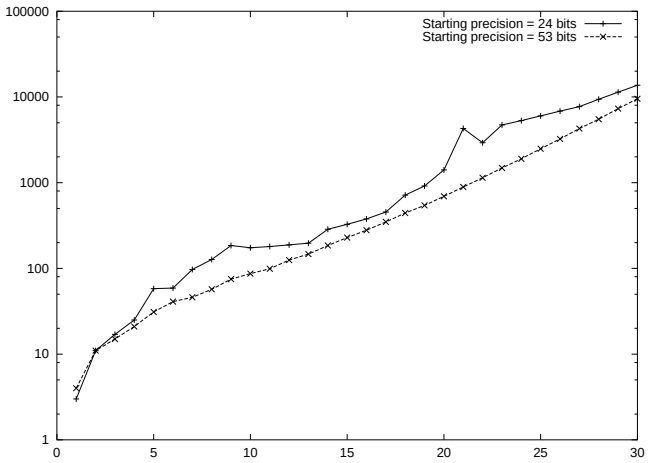

Figure 2: Number of examined intervals (in logarithmic scale) with respect to the degree and the starting computing precision.

The experiments presented here illustrate the behaviour of this algorithm for some quite difficult problems, even if they imply only polynomials.

The first series concerns the Chebychev polynomials which can be defined for instance by:

$$C_n(\cos\theta) = \cos n\theta.$$

The degree of C_n is n. They are known to be difficult to evaluate accurately even if they take their values in $[-1, 1]$, because their coefficients are large. A consequence is thus that it is quite difficult to get a small "residual" $F(X)$, smaller than the stopping threshold ε_Y. Figure 2 presents for each Chebychev polynomial of degree from 1 to 30 the total number of examined intervals (with a logarithmic scale) for two different initial computing precisions: 24 bits (which corresponds to the IEEE single precision) and 53 bits (which corresponds to the IEEE double precision). The starting search interval is $[-2, 2]$ and $\varepsilon_X = \varepsilon_Y = 10^{-6}$, which is the relative precision achievable in single precision. The benefit of using a multiple precision arithmetic appears clearly: the polynomial evaluations are tighter and thus intervals are rejected earlier when a high precision is used than with a low precision. Figure 3 illustrates the power of interval Newton iteration: again, the starting search interval is $[-2, 2]$, $\varepsilon_X = \varepsilon_Y = 10^{-6}$ and the starting precision is 24 bits. At most two doublings of the precision occurred. The results are very satisfactory since no superfluous intervals are given in the list of potential results and the existence (and uniqueness) of the roots in each isolating interval is proven for most of the intervals.

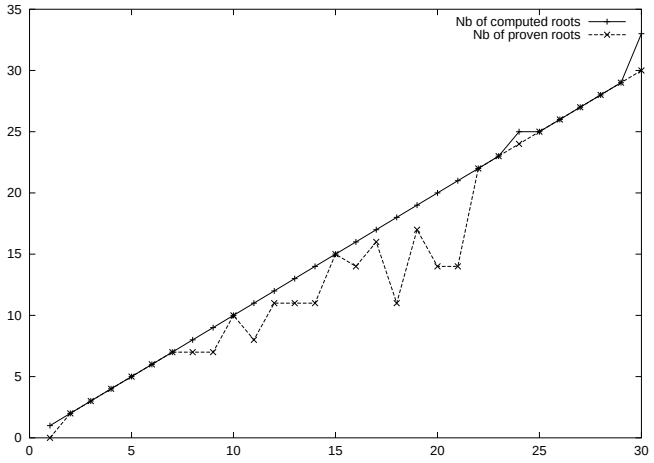

Figure 3: Number of computed roots and roots with proven existence.

A second series presents quite the same conclusions obtained with the Wilkinson polynomials $W_n(x) = \prod_{i=1}^{n}(x-i)$ written in the expanded form. The initial precision is chosen large enough to enable the exact representation of the coefficients. These polynomials are difficult to evaluate precisely even in multiple precision "scalar" arithmetic; this can easily be checked thanks to the forward error bound given in [8]:

$$|p(x) - \widehat{q_0}| \leq 2n\varepsilon \sum_{i=0}^{n} |a_i|.|x|^i$$

where $p(x) = \sum_{i=0}^{n} a_i x^i$ is the polynomial to evaluate and $\widehat{q_0}$ is the value computed by Horner's rule. Since the coefficients of W_n are of the order of magnitude of $n!$, the assertion concerning the bad quality of the evaluation of Wilkinson polynomials is obtained. The results confirm the difficulty of accurately evaluating a Wilkinson polynomial: the required precision on the range (called ε_Y in our algorithm) cannot be made much smaller than $\mathcal{O}(n!)$. Consequently it is very difficult for our algorithm (essentially very time-consuming) to discard intervals not containing zero. The results are thus small enclosures for the roots along with a proof of their existence and uniqueness and a long list of other intervals, covering almost the whole interval $[1, n]$.

For $n = 20$ and with the coefficient of X^{19} perturbed by the interval $[-2^{-19}, 2^{-19}]$, every point between 8 and 20 is a root of a perturbed polynomial belonging to this interval polynomial; indeed, our algorithm returns small enclosures for the roots 1 to 7 and a covering of $[7.91, 22.11]$.

Lastly, a polynomial having a double root is tested [9]. It is known that for a root of multiplicity μ the backward error is of order $\varepsilon^{1/\mu}$, or roughly speaking that in order to get an error of order ε_X on the root, an error ε_X^{μ} must be obtained for the polynomial evaluation. This intuitive fact is experimentally confirmed. Moreover, interval Newton iteration even enables to prove that the decimal to binary coefficient conversion with rounding to nearest has transformed the multiple root into two close single roots.

6 Conclusion and future work

In this paper we have presented an algorithm for the determination of real zeros of univariate equations. This algorithm is based on interval arithmetic in order to provide guaranteed enclosures of every zero in the initial search domain and it also benefits from multiple precision arithmetic which enables it to compute arbitrarily accurate enclosures. Its convergence towards arbitrarily tight enclosures and its termination property have been proven. Experimental results illustrate the behaviour of this algorithm on some difficult problems and the results are conform to what was expected.

The next step of this work consists in the study and implementation of an algorithm solving linear systems. It will enable us to implement a multivariate Newton method.

References

[1] G. Alefeld and J. Herzberger. *Introduction to interval analysis.* Academic Press, 1983.

[2] R. Baker Kearfott. *Rigorous global search: continuous problems.* Kluwer, 1996.

[3] R. Baker Kearfott. Empirical evaluation of innovations in interval branch and bound algorithms for nonlinear systems. *SIAM J. Sci. Comput.*, 18(2):574–594, March 1997.

[4] R. Baker Kearfott and G.W. Walster. On stopping criteria in verified nonlinear systems or optimization algorithms. *ACM TOMS*, 26(3):373–389, September 2000.

[5] CANT Research Group. Arithmos: a reliable integrated computational environment. University of Antwerpen, Belgium, `http://win-www.uia.ac.be/u/cant/arithmos`, 2001.

[6] E. Hansen. *Global optimization using interval analysis.* Marcel Dekker, 1992.

[7] E. Hansen and R.I. Greenberg. An interval Newton method. *J. of Applied Math. and Computing*, 12:89–98, 1983.

[8] N. Higham. *Accuracy and stability of numerical algorithms.* SIAM, 1996.

[9] P. Langlois and N. Revol. Validating polynomial numerical computations with complementary automatic methods. Research report (submitted to

Math. and Comp. in Sim.) 2001-18, LIP, Ecole Normale Superieure de Lyon, `http://www.ens-lyon.fr/LIP/Pub/rr2001.html`, 2001.

[10] G. Mayer. Epsilon-inflation in verification algorithms. *Journal of Computational and Applied Mathematics*, 60:147–169, 1995.

[11] A. Neumaier. *Interval methods for systems of equations.* Cambridge University Press, 1990.

[12] H. Ratschek and J. Rokne. *New computer methods for global optimization.* Ellis Horwood Ltd, 1988.

[13] G.W. Stewart. What is rounding error? `http://www.cs.umd.edu/~stewart`, in `/pub/misc`, 2000.

[14] P. Van Hentenryck, D. Macallester, and D. Kapur. Solving polynomial systems using a branch and prune approach. *SIAM J. Numer. Anal.*, 34(2):797–827, April 1997.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399